OPTIMIZING SPAM FILTERING WITH MACHINE LEARNING

TEAM ID: NM2023TMID17659

TEAM SIZE : 04

TEAM LEADER : GOVINDASAMY M (20UCS4712)

TEAM MEMBER : 1. GOVINDHARAJ B (20UCS4713)

2. VISHWANATHAN T (20UCS4741)

3. VIGNESH V (20UCS4740)

OPTIMIZING SPAM FILTERING WITH MACHINE LEARNING

1. INTRODUCTION :

Spam messages refer to unsolicited or unwanted messages/emails that are sent in bulk to users. In most messaging/emailing services, messages are detected as spam automatically so that these messages do not unnecessarily flood the users' inboxes. These messages are usually promotional and peculiar in nature. Thus, it is possible for us to build ML/DL models that can detect Spam messages.

OVERVIEW :

A spam filter is a program used to detect unsolicited, unwanted and virus-infected emails and prevent those messages from getting to a user's inbox. Like other types of filtering programs, a spam filter looks for specific criteria on which to base its judgments.

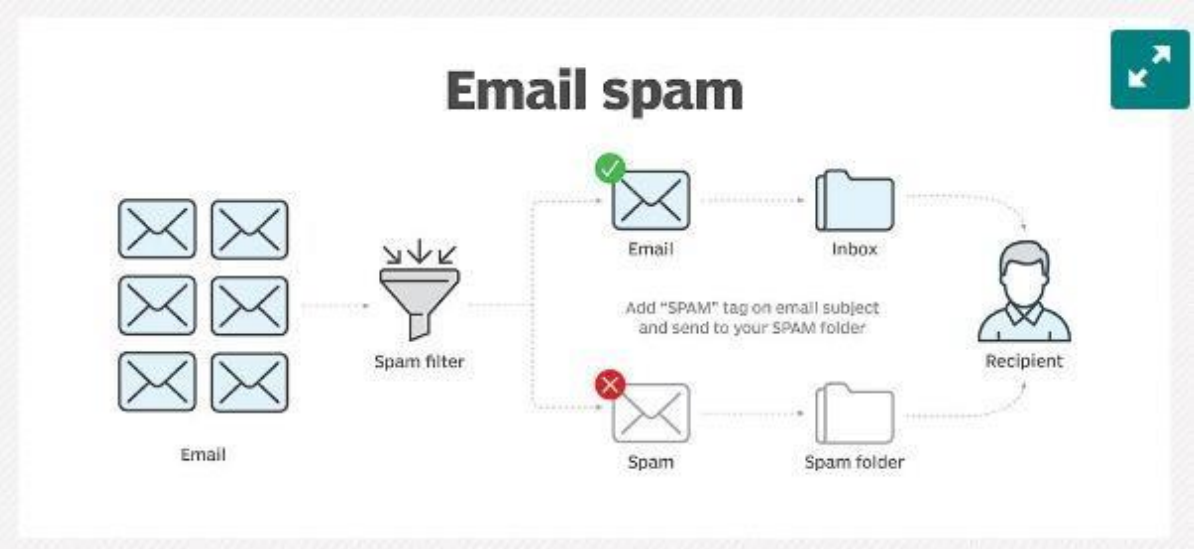
Internet service providers (ISPs), free online email services and businesses use email spam filtering tools to minimize the risk of distributing spam. For example, one of the simplest and earliest versions of spam filtering, like the one that was used by Microsoft's Hotmail, was set to watch out for particular words in the subject lines of messages. An email was excluded from the user's inbox whenever the filter recognized one of the specified words.

This method is not especially effective and often omits perfectly legitimate messages, called *false positives*, while letting actual spam messages through.

More sophisticated programs, such as Bayesian filters and other heuristic filters, identify spam messages by recognizing suspicious word patterns or word frequency. They do this by learning the user's preferences based on the emails marked as spam. The spam software then creates rules and applies them to future emails that target the user's inbox.

For example, whenever users mark emails from a specific sender as spam, the Bayesian filter recognizes the pattern and automatically moves future emails from that sender to the spam folder.

ISPs apply spam filters to both inbound and outbound emails. However, small to medium enterprises usually focus on inbound filters to protect their network. There are also many different spam filtering solutions available. They can be hosted in the cloud, hosted on servers or integrated into email software, such as Microsoft Outlook.



PURPOSE :

Spam filters are algorithms that detect unsolicited, undesired or infected emails, and block those messages from reaching inboxes. There are a variety of spam filters, with detection capabilities ranging from basic pattern matching, all the way through to machine learning.

Spam filters do a lot of work. And much of it is complex, confusing and constantly changing. With the pressure to keep viruses out and determine what's unsolicited vs solicited, it's no wonder they sometimes don't get everything right.

Don't worry though, there's lots you can do to ensure you're playing by the rules and feeding spam filters the information they need, to make an informed decision about your email. More on the steps you can take later.

There are three main types of spam filters: Email service provider, third-party and desktop.

1. Email service provider spam filters

These are the spam filters you're likely using without even knowing. Email service providers are the cloud based services that most people use to send, receive and organize their emails. Think Gmail, Yahoo Mail, Office 365 and Hotmail (yes, Hotmail is still a thing!).

While you may think of those names being mainly used by individuals, many of them offer services packaged for companies to use — en masse with their employees — with one popular service being Google Workspace.

These email service providers have their own built-in spam filters with advanced algorithms that look for the latest spam patterns and apply machine learning (ML) to evaluate a sender's trust and the message's inbox-worthiness.

2. Third-party spam filters

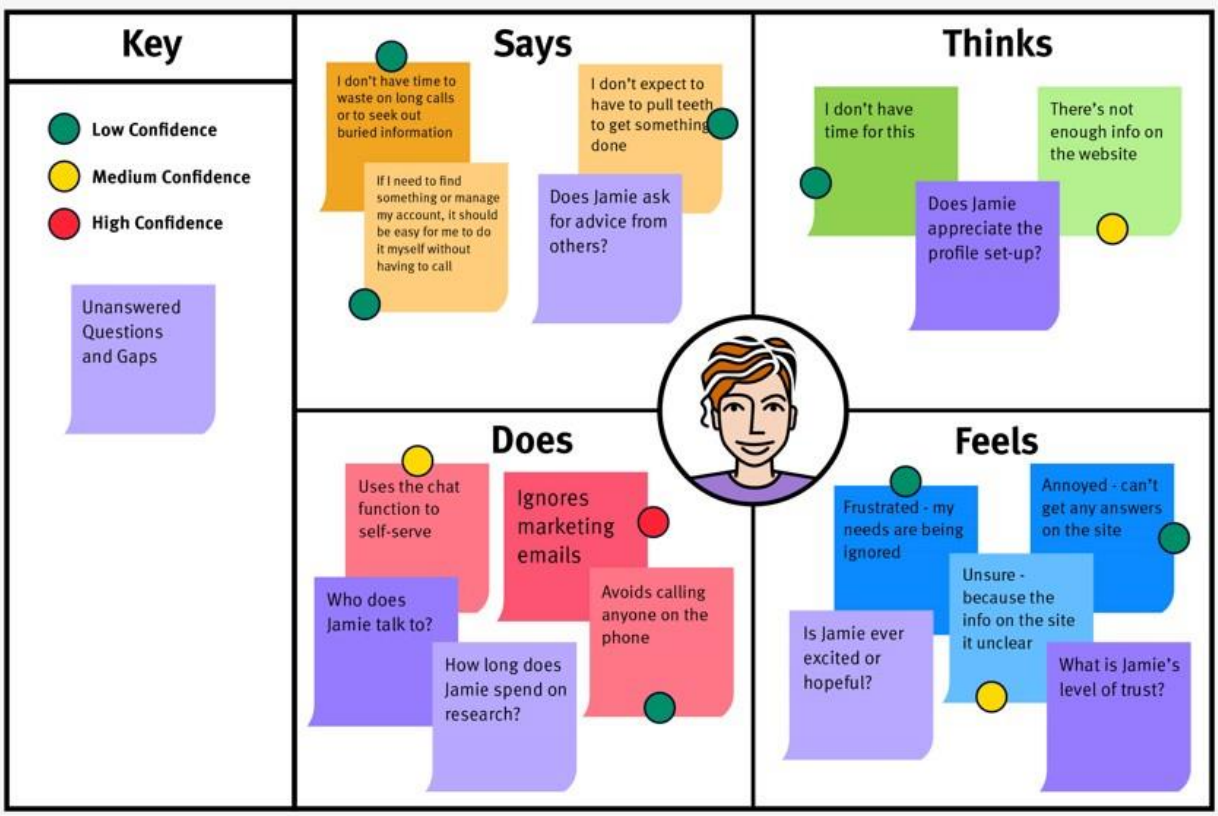
Third party spam filters are additional layers of spam protection that companies add to their email system (whether it's onsite or hosted in the cloud). They work hand-in-hand with the company's email infrastructure to provide a higher degree of fine-tuning, with the ability, for example, to choose between aggressive anti-spam measures (letting fewer emails in), and looser measures (letting more emails in).

Companies that employ third party spam filters also benefit from more robust reporting and analytics so they can better understand email and spam activity within the company. This can be useful for proactively thwarting targeted phishing attacks against a company.

3. Desktop spam filters

Desktop spam filters are pieces of software that you purchase and install on your computer. They monitor messages once they make it past your email service provider's spam filters and add an extra layer of protection. The level of protection can be configured to specifically suit your spam tolerance and tastes. As the filters built into email service providers become more and more sophisticated, desktop spam filters have become less popular.

2. PROBLEM DEFINITION AND DESIGN THINKING
EMPATHY MAP



IDEATION AND BRAINSTROMING :

3. RESULT :


127.0.0.1:5000
SmartBridge
Home
About
Spam
SMS SPAM Detection
Using NLP.


127.0.0.1:5000/#about
SPAM DETECTION
Spam Detector for Short Message Service (SMS).
127.0.0.1:5000/#about

4. ADVANTAGES AND DISADVANTAGES

Study ID	Advantages	Disadvantages
S1 [1]	Combination of machine learning algorithms with user generated features	Users need to select features manually
S2 [2]		No classification algorithm is used
S3 [3]	Classification based on two algorithms	Threshold selection
S4 [5]	Combination of client and server side algorithms	Challenge-response technique suffers from server side traffic and user interaction problems
S5 [10]	Accurate as Naïve Bayesian with necessary feature extraction	Complex implementation
S6 [11]	Incorporating Apriori Algorithm	
S7 [12]	Used a weighting Mechanism to reduce false negatives	
S8 [7]		Suffers from implementation complexity
S9 [13]	Combination of two formulas gives better result in terms of false positives	
S10 [14]	Concludes SVM outperforms other algorithms and created a baseline for further comparison	
S11 [15]	Although SVM gives better results in Spam identification Bayesian is more feasible for mobile applications	Extensive feature engineering is needed for better accuracy
S12 [16]	Demonstrates the need of spam filtering in spite of having established email spam filtering	
S14 [18]	Lightweight and focuses on runtime	

5. APPLICATIONS :

A spam filtering solution cannot be 100 percent effective. However, a business email system without spam filtering is highly vulnerable, if not unusable. It is important to stop as much spam as you can, to protect your network from the many possible risks: viruses, phishing attacks, compromised web links and other malicious content.

Spam filters also protect your servers from being overloaded with non-essential emails, and the worse problem of being infected with spam software that may turn them into spam servers themselves.

By preventing spam email from reaching your employees' mailboxes, spam filters give an additional layer of protection to your users, your network, and your business.

6. CONCLUSION

In the last two decades, spam detection and filtration gained the attention of a sizeable research community. The reason for a lot of research in this area is its costly and massive effect in many situations like consumer behavior and fake reviews. The survey covers various machine learning techniques and models that the various researchers have proposed to detect and filter spam in emails and IoT platforms. The study categorized them as supervised, unsupervised, reinforcement learning, etc. The study compares these approaches and provides a summary of learned lessons from each category. This study concludes that most of the proposed email and IoT spam detection methods are based on supervised machine learning techniques. A labeled dataset for the supervised model training is a crucial and time-consuming task. Supervised learning algorithms SVM and Naïve Bayes outperform other models in spam detection. The study provides comprehensive insights of these algorithms and some future research directions for email spam detection and filtering.

7. FUTURE SCOPE

Efficient pattern detection in spam mail filtering plays crucial role. Using RFD model spam detection gives the spam patterns, non –spam patterns and general patterns which easily identify the whether the mail is spam or ham. The current method which uses the pattern detection method does not include the general patterns. RFD gives the general patterns of which user can decide to determine whether he wants to put the mail as spam or non-spam to avoid the loss of important mails. The images which are in forms of spams are also detected using File Properties, Histogram and Hough Transform. The current proposed system is for English language mails but as future scope we can design the system for multiple languages.

8. APPENDIX

```
import numpy as np
    import pandas as pd
    import matplotlib.pyplot as plt
    import tensorflow as tf

    # For spliting dataset
    from sklearn.model_selection import train_test_split

    # For preprocessing data
    import re
    import nltk
    import string
    from tensorflow.keras.preprocessing.text import
  Tokenizer
    from tensorflow.keras.preprocessing.sequence import
  pad_sequences

    # For building the model
    from tensorflow.keras.callbacks import EarlyStopping
    from tensorflow.keras.models import Sequential
    from tensorflow.keras.layers import (
        Dense, Embedding, Dropout,
        GlobalAveragePooling1D, Input
    )
    from tensorflow.keras.models import Model

    SEED = 42
    tf.random.set_seed(SEED)
    np.random.seed(SEED)

    data_path = "spam.csv"

    df = pd.read_csv(data_path, encoding='ISO-8859-1')
    df = df[['v1', 'v2']]
```

```
    df.columns = ['Category', 'Content']
    df.head()

    plt.figure(figsize=(8, 6))
    df.Category.value_counts().plot(kind='bar',
  color='green')

    stopwords = nltk.corpus.stopwords.words('english')
    ps = nltk.PorterStemmer()

    def clean_text(text):
        # Remove punctuation from text
        text = "".join([word.lower() for word in text if
  word not in string.punctuation])
        tokens = re.split('\W+', text)
        # Get Origin of words (books -> book)
        text = " ".join([ps.stem(word) for word in tokens
  if word not in stopwords])
        return text

        df['Clean_text'] = df['Content'].apply(lambda x:
  clean_text(x))
    df.head()

    # {ham:0, spam:1}
    df['label'] =
  df['Category'].astype('category').cat.codes
    df.head()

    X_train,X_test,y_train,y_test=train_test_split(

  df['Clean_text'],df['label'],test_size=0.25,random_state=SEE
  D)

    max_len=50 # max words in every input string
```

```
    trunc_type='post' # if number of words are more than max_len then trunct from the last of the string
    padding_type='post' # if number of words are less than max_len then add
    oov_token_1='<OOV>'# out of vocabulary token
    vocab_size=500

    tokenizer=Tokenizer(num_words=vocab_size,char_level=False,oov_token=oov_token_1)
    tokenizer.fit_on_texts(X_train)

    print(f"Total token : {len(tokenizer.word_index)}")
    for token_word in tokenizer.word_index:
        print(f"{token_word}: {tokenizer.word_index[token_word]}")
        if tokenizer.word_index[token_word] >10:
            break

    X_train = tokenizer.texts_to_sequences(X_train)
    X_train = pad_sequences(X_train,maxlen=max_len,padding=padding_type,truncating=trunc_type)
    X_test=tokenizer.texts_to_sequences(X_test)
    X_test=pad_sequences(X_test,maxlen=max_len,padding=padding_type,truncating=trunc_type)
    X_train.shape, X_test.shape

    input_shape = X_train.shape[1:]
    input_shape

    input = Input(shape=input_shape)
    x = Embedding(input_dim=vocab_size, output_dim=12)(input)
    x = GlobalAveragePooling1D()(x)
    x = Dense(24, activation='relu')(x)
    x = Dropout(0.2)(x)
```

```
    output = Dense(1, activation='sigmoid')(x)
    model = Model(inputs=input, outputs=output)
    model.compile(optimizer = 'adam', loss =
  'binary_crossentropy', metrics=['accuracy'])

    model.summary()

    tf.keras.utils.plot_model(model)

    num_epochs=50
    # If val_loss increases five times it will stop model
  fitting
    early_stop=EarlyStopping(monitor='val_loss',patience=5)
    history=model.fit(
        X_train,
        y_train,
        epochs=num_epochs,
        validation_data=(X_test,y_test),
        callbacks=[early_stop]
    )

    model.evaluate(X_test, y_test)

    def plot_loss_curves(history):
      loss = history.history['loss']
      val_loss = history.history['val_loss']

      accuracy = history.history['accuracy']
      val_accuracy = history.history['val_accuracy']

      epochs = range(len(history.history['loss']))

      # Plot loss
      plt.plot(epochs, loss, label='training_loss')
      plt.plot(epochs, val_loss, label='val_loss')
      plt.title('Loss')
```

```
131.      plt.xlabel('Epochs')
132.      plt.legend()
133.
134.      # Plot accuracy
135.      plt.figure()
136.      plt.plot(epochs, accuracy, label='training_accuracy')
137.      plt.plot(epochs, val_accuracy, label='val_accuracy')
138.      plt.title('Accuracy')
139.      plt.xlabel('Epochs')
140.      plt.legend();
141.
142.      plot_loss_curves(history)
143.
```